02.14.00

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Harold E. Helson

Serial No.: To Be Assigned

Filed: Herewith (This application claims the benefit of U.S. Provisional Application Serial No. 60/119,654 entitled STRUCTURE DIAGRAM GENERATION, filed on February 11, 1999.)

Title: ENHANCING STRUCTURE DIAGRAM GENERATION

Box Patent Application
Assistant Commissioner for Patents
Washington, DC 20231

TRANSMITTAL LETTER

Dear Sir:

Enclosed for filing in the above-referenced patent application are the following documents:

1. New U.S. Patent Application entitled **ENHANCING STRUCTURE DIAGRAM GENERATION**

   and naming as inventor(s): Harold E. Helson

   the Application including 44 pages comprising:

   28 pages of specification including:
   6 pages of claims (claims 1-12) and;
   1 page of abstract; and
   16 pages of informal drawings (Figures 1A to 16).

2. Declaration and Power of Attorney (unexecuted).

3. Source Code Appendix with Cover Sheet For Source Code Appendix.

4. Postcard.

The Commissioner is hereby **not authorized** to charge the filing fees to our Deposit Account No. 08-0219.

Respectfully submitted,

Dated: February 11, 2000

Jason A. Reyes
Registration No. 41,513
Attorney for Applicant

Hale and Dorr LLP
60 State Street
Boston, MA 02109
Tel.: (617) 526-6010
Fax: (617) 526-5000

Attorney Docket No. 103544.127

EXPRESS MAIL LABEL NO. EM259723534US
DATE OF DEPOSIT February 11, 2000

ENHANCING STRUCTURE DIAGRAM GENERATION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of United States Provisional Application Serial No. 60/119,654 entitled STRUCTURE DIAGRAM GENERATION filed on February 11, 1999, incorporated herein.

REFERENCE TO SOURCE CODE APPENDIX

A source code appendix forms part of this application. The appendix, which includes a source code listing relating to an embodiment of the invention, includes 30 pages.

Background of the Invention

This application relates to enhancing structure diagram generation.

A molecule is typically represented in a computer by a connection table that identifies atoms in the molecule and specifies connections ("bonds") among the identified atoms. The connection table may also describe associated

properties such as atom type, bond order, charge, and stereochemistry. A diagrammatic representation of the molecule may be derived from the connection table. Examples of a connection table and a corresponding diagram are illustrated in Figs. 1A-1B (for clarity, hydrogen atoms are not shown).

In chemistry, with reference to Fig. 2, a chain of atoms that closes on itself is known as a ring. In the context of a ring or a ring system (see below), a bridge is a chain of atoms that begins at an origin point (which is an atom) in the ring or system, and connects back to the same ring or system at least two atoms away from the origin point, to form an additional ring. A chain that reconnects at the same origin point instead is known as "spiro". A chain that reconnects to an atom that is adjacent to the origin point is known as "fused".

A ring system, which is also known as a "cyclic system", is a group of rings such that (1) each ring shares one or more bonds with another ring in the group and (2) the group cannot be divided into smaller cyclic systems. An arrangement in which two rings are connected by a linking, non-cyclic ("acyclic") bond is considered to include two cyclic systems, not one. As used herein, "ring system" has a meaning consistent with an understanding that a spiro ring includes two distinct ring systems.

Summary of the Invention

A method and a system are provided for enhancing structure diagram generation ("SDG"). In SDG, aesthetic two-dimensional ("2-D") coordinates for use in a diagrammatic representation ("diagram") of a molecule are derived from a connection table for the molecule. SDG may also improve the aesthetic qualities of a chemical structure diagram having existing coordinates, if available. SDG is enhanced by expressing the symmetry present in the molecule, by making use of symmetry in the 2-D dynamics used to lay out rings and chains, by construction of bridges using an open polygon method together with a potential function, and by an elegant approach to the relative positioning of molecules ("free rectangle method").

Other features and advantages will become apparent from the following description, including the drawings, and from the claims.

Brief Description of the Drawings

Fig. 1A is an illustration of computer data.

Figs. 1B-1C, 3-4, 6-13 are illustrations of output produced by software.

Fig. 2 is an illustration of chemical structures.

Figs. 5, 14-16 are flow diagrams of computer-based procedures.

Detailed Description

Structure diagram generation ("SDG") is a process in which two dimensional ("2-D") coordinates are derived from a connection table for a structure, allowing a diagram of the molecule to be displayed or printed. SDG is described in detail in H. E. Helson, "Structure Diagram Generation", in "Reviews in Computational Chemistry", K. B. Lipkowitz and D. B. Boyd, Eds., Wiley-VCH, New York, 1999, Vol. 13, at 313-398, which is incorporated herein. This application is filed simultaneously with a United States patent application entitled DERIVING CHEMICAL STRUCTURAL INFORMATION, serial no. _____, which is incorporated herein.

The coordinates may be derived with or without preexisting coordinates. Cases without preexisting coordinates ("de novo" cases) are common and include chemical name translation, isomer enumeration, translation from a linear notation such as SMILES, nickname/superatom expansion, and automated structure elucidation.

In cases in which preexisting coordinates are available ("structure cleanup" cases), it may be possible to improve a structure diagram while preserving some or all existing stylistic choices. For example, if a structure diagram is drawn with or imported into a structure drawing program, the

program may be directed to "clean up" undesirable aspects of the structure diagram. In another example, diagram improvements may be needed in the case of a synthesis planning program, in which structure diagrams are generally well drawn but may have had bonds broken and reformed in awkward locations.

SDG may also be needed in conversions of structure diagrams from three-dimensional ("3-D") to 2-D. In at least some cases, structure diagrams that are stored and manipulated in a 3-D form may be converted to 2-D diagrams upon display to make the structure diagrams more easily recognizable to human users.

As a result, a connection table to which SDG is applied may have 2-D or 3-D coordinates or may lack coordinates.

SDG includes at least four possible stages ("phases"): perception, pre-assembly analysis, assembly, and post-assembly. The pre-assembly phase, if applicable, may include deriving a feature such as the shape of a ring system that is subsequently attached whole to an acyclic portion in the assembly phase. In the assembly phase, the neighbors of an atom that has been positioned (a "seed" atom) are each examined in turn, and are positioned at respective aesthetic angles and distances from the seed atom. Figs. 1A-1C illustrate an example. A connection table of a simple molecule (Fig. 1B) is shown in Fig. 1A.

In a specific embodiment, one of the atoms is arbitrarily chosen as a first seed atom. The neighbors of the first seed atom are positioned; each of the neighbors in turn takes the role of the seed atom in subsequent iterations, until all of the atoms are positioned, as shown in Fig. 1C for the connection table of Fig. 1A.

SDG is enhanced as described below.

In a first aspect of the enhancement, symmetry is used in the assembly phase, i.e., for general layout. Chemical structure diagrams that express molecular symmetry facilitate human interpretation of the chemical structures that are represented. For example, the presence of symmetry provides clues for the molecular substance's synthesis. Symmetry affects the substance's physical properties (particularly those affected by entropy), such as melting and boiling points, and heat of vaporization. Symmetry can affect the substance's light-bending properties. In particular, a substance that has a plane of symmetry is not "optically active". In general, since the human eye tends to recognize symmetry quickly, a diagram that expresses a molecule's symmetry allows the symmetrical characteristics of the molecule to be rapidly perceived by a human viewer.

According to the enhancement, when a diagram is to be produced for a molecule, symmetry inherent in the molecule is perceived, and during layout of

the structure diagram, representations of atoms and bonds are positioned to express the perceived symmetry. In a specific implementation, a plane of symmetry (also known as a mirror plane) perceived in a molecule is expressed vertically or horizontally (see Fig. 3).

In a first step in using symmetry in general layout (see Figs. 4 and 14), an instance of symmetry is determined (step 1010). Detection of symmetry is described in M. Razinger, K. Balasubramanian, and M. E. Munk, "Graph Automorphism Perception Algorithms in Computer-Enhanced Structure Elucidation", *J. Chem. Inf. Comput. Sci.*, **33**, 197 (1993). The instance of symmetry may be based on one or more of rotation, reflection (see b. in Fig. 4), inversion, and translation, and may or may not take stereochemistry into account. A particularly effective combination in the determination is an instance of symmetry based on rotation and reflection, and a flexible incorporation of stereochemistry. If a full consideration of stereochemistry does not reveal an instance of symmetry, a partial consideration of stereochemistry, e.g., of double bond stereochemistry only, is employed. The instance of symmetry is here represented as a list of orbits, i.e., a list of groups of equivalent atoms and bonds.

Additionally, a "pivot" point is determined for each orbit (step 1020). The pivot point is determined to be the one or more atoms or bonds that resides at the graph-theoretic center of the atoms and bonds in the orbit, i.e., those atoms (bonds) having the smallest value of the largest graph-theoretic distance to any other atom (bond) in the orbit. The graph-theoretic distance between two atoms (bonds) is equal to the number of bonds (atoms) in the shortest path between them. For example, in Figs. 3-4, the nitrogen atom is determined to be the pivot point; in n-butane, the central bond is determined to be the pivot point, and in 1,2,3-trimethylcyclopropane, all cyclic atoms and bonds are determined to be the pivot point.

The "order" of each orbit for each instance of symmetry is also determined (step 1030). The order indicates whether the instance corresponds to a two-fold rotation, a three-fold rotation, a four-fold rotation, and so on, or a reflection. In cases in which the symmetry of an orbit includes both reflection and an N-fold rotation, N being greater than 2, it is advantageous to treat the instance as having an order indicating that the instance corresponds to the N-fold rotation. Thus, rotational symmetry takes priority over reflection if the associated rotation is at least three-fold.

When an atom or bond is positioned during the assembly phase (step 1040) (see Figs. 1, 4), attention is paid to whether the atom or bond belongs to one of the determined instances of symmetry. In a case in which the atom or bond so belongs, after the atom or bond is positioned, other atoms or bonds, respectively, that belong to the same instance are positioned immediately thereafter (step 1050) (see Figs. 4-5). If the type of symmetry involved is reflection (see Fig. 5), the other atom or bond is placed on the opposite side of the mirror line that runs through the pivot point of the group in the instance. In such cases, the direction may be arbitrary if only two atoms have been placed. If the type of symmetry involved is rotation, the symmetrically equivalent atoms or bonds are positioned at appropriate rotational points, based on the pivot point. First positioning an atom or bond that represents the pivot point facilitates symmetric positioning but is not always possible, such as when multiple regions of independent symmetry are involved (e.g., in an unsymmetrical ether, each end of which is locally symmetric).

After all atoms and bonds have been placed, the structure diagram is rotated so that its mirror plane is horizontal or vertical (step 1060) (see Fig. 3).

In another aspect of the enhancement of SDG, symmetry is used in a "dynamics" method of layout. A 2-D version of molecular dynamics is used in

some situations to lay out structure diagrams of molecules in connection with designing new ring systems, improving existing ring systems, or laying out or improving acyclic portions. Such an effort may use a predefined set of optimal bond lengths and angles ("parameters"), or may seek to equalize adjacent lengths and angles. The process is iterative, wherein in each iteration the difference between a current parameter and an optimal parameter is calculated for each atom and bond, and is interpreted as a corrective force on the atom or bond, which affects the position of the atom or bond as submitted to the next iteration. The iterative process continues until the net corrective force on every atom or bond is zero or nearly zero, so that the structure diagram for the molecule is determined to be at equilibrium.

A method of adding symmetry as a parameter in dynamic ring layout is now described (Fig. 15). The concepts presented are also applicable to acyclic systems. Dynamic ring layout in general is described in H. E. Helson, Ph.D., Thesis, "Simulation of Carbene Chemistry and Other Problems in Computer-Assisted Organic Synthesis.", Purdue University, 1993; H. E. Helson and W. L. Jorgensen, J. Chem. Inf. Comput. Sci., 34, 962 (1994), "Computer-Assisted Mechanistic Evaluation of Organic Reactions. 25. Structure Diagram Positioning"; and H. E. Helson, "Structure Diagram Generation", in "Reviews in

Computational Chemistry", K. B. Lipkowitz and D. B. Boyd, Eds., Wiley-VCH, New York, 1999, Vol. 13, at 313-398. As shown below, a symmetry term is incorporated in a force field, which drives symmetrically equivalent regions in different parts of the structure diagram toward a common appearance. Instances of symmetry are determined (step 2010). The symmetry detection referenced above is an example of such a determination. In a specific implementation, the instances are determined based on rotation and reflection, without regard for bond orders or types, atom characteristics (e.g., mass, type, charge), or acyclic portions, and the determination focuses exclusively on the locations of the bonds. The instances of symmetry may be determined without supplied coordinates. The determination takes double bond isomerism into account: E and Z isomers are recognized as not being equivalent. Specific implementations may also take into account 2-D graphics-based characteristics not normally connected with molecular symmetry, such as bond zig-zags, or whether a bond is "exterior" or "interior", i.e., whether or not the bond has a clear path to the edge of the drawing area.

The instance of symmetry, regardless of character and origin, may be represented in any of several ways. In a specific implementation, the instance is represented by two lists of groups: a list of equivalent triplets of atoms, and a

list of equivalent pairs of bonds (see Fig. 6, in which the top and bottom sequences illustrate equivalent bonds and atom triplets, respectively, and each dot marks the central atom in a triplet).

In each iteration for each triplet, a respective force term ("F_a") is added for the atom in the center of the triplet (step 2020). An optimal interior angle ("optimal angle") of the triplet of atoms is derived, as the average of the interior angles of all the triplets in an orbit, i.e., in a group of symmetrically equivalent atoms or bonds. F_a is based on, and in a specific implementation is proportional to, the difference between the optimal angle and the current angle. F_a acts along the angle's bisector, in a direction that would bring the angle closer to the optimal angle. F_a may compete with other terms, such as a bond angle term for equalizing adjacent bond angles.

In each iteration, another respective force term ("F_b") is added for each symmetric bond (step 2030). F_b has the effect of lengthening or shortening a bond to make the bond's length more similar to the lengths of the other bonds in the orbit. A bond's length is changed by moving the atoms at the bond's endpoints closer together or farther apart. Thus F_b is expressed by treating F_b as a force on each of its two adjacent atoms. F_b may compete with other terms, such as a bond length term for equalizing adjacent bond lengths.

During each iteration, a net force on each atom is calculated, as the sum of the forces including F_a and F_b acting on the atom (step 2040). The position of each atom is moved by an amount proportional to the respective net force. In a specific implementation, the iterative process is determined to be complete when the largest net force to be accounted for in the iteration is smaller than a specified threshold size (step 2050).

Fig. 7 illustrates an example of net forces and the iterative evolution. In Fig. 7, double arrows show the forces due to symmetry on selected atoms and bonds, single dashed arrows represent bond angle forces re-expressed as atom translation, and other forces not related to symmetry are omitted for clarity and to reduce clutter. The rightmost structure diagram in Fig. 7 represents an improvement over the leftmost structure diagram.

Construction of bridged cyclic systems may involve problems of atom and bond overlap, and irregular angles. In another aspect of the enhancement of SDG, bridges in cyclic systems are constructed using an open polygon method in conjunction with a potential function. In an example illustrated in Fig. 8, SDG has already produced two rings that are part of a tricyclic system. In a regular polygon method, a third ring (indicated by dashed lines in Fig. 8) is attached by constructing the third ring as a regular polygon, so as to fuse the third ring to

either one of the rings already produced. In such a case, as shown in the top two example sequences in Fig. 8, uneven coordinates are produced. Alternatively in the regular polygon method, the regular polygon can be attached directly at the two bridgehead atoms, as shown in the bottom example sequence in Fig. 8, but uneven results are still achieved. By contrast, the open polygon method is able to generate evenly spaced coordinates between the two termini where the ring will be attached, as shown in an example sequence in Fig. 9, in which a five-membered ring is fused onto a bicyclo[6.1.0] system. See H. E. Helson, "Structure Diagram Generation", in "Reviews in Computational Chemistry", K. B. Lipkowitz and D. B. Boyd, Eds., Wiley-VCH, New York, 1999, Vol. 13, at 313-398, which is incorporated herein.

In the open polygon method, coordinates of missing points are derived from two grounding points, the number of missing points, and an optimal bond length ("*d*"), such that, as shown in Fig. 9, interior angles ("β") at the two grounding points are equal and the remaining interior bond angles ("α") are all equal.

The open polygon method can be used to create bridges. In Fig. 9, the two grounding points correspond to the two bridgehead atoms. In at least some cases, however, the resulting bridge may be determined to be too close or too far

away from the base ring skeleton such that the resulting bridge crowds the base ring skeleton. As a result, the circumference of the bridge is varied by varying the value of *d* in Fig. 9.

Fig. 10 illustrates an example of an application of the open polygon method to bridge construction. The leftmost structure diagram represents the preexisting ring skeleton to which the bridge will be affixed. The other structure diagrams represent the results of applying the open polygon method using various values of *d*. The one producing the least congestion, among other factors, is chosen. More specifically, the value of *d* that is selected is the value that (1) produces the least congestion between the bridge and the base ring skeleton, as measured, for example, by a two-body inverse-distance squared potential function, (2) does not produce near-linear bond angles, i.e., does not produce an α that is nearly 180 degrees, and (3) uses a bond length *d* that is close to the optimal bond length, as may be expressed in a weighted function, such as:

$$\text{Rating} = c1 * \text{Congestion} + c_2 * \max(0, (180 - \alpha) - \text{threshold}) + c_3 * | \textit{scale} - 1.0 |$$

In such a function, c_1, c_2 and c_3 are constants determined in a specific implementation; and *scale* is the ratio of d to the standard bond length. In a specific implementation, the bond angle term is active only above a certain threshold, such as 120 degrees. The version of the bridge that minimizes the rating is chosen.

Fig. 11 illustrates an application of the ratings method. Each line starting with "Rating for" indicates a rating computed for a particular bond length. For example, the second such line reports a rating of 313.185 for a bond length scale of 0.5 where the contribution for congestion is 45.185, the contribution for a non-unitary bond length is 40.00, and the contribution for a non-linear bond angle of 177 is 228. With respect to Fig. 11, the bond length scale that achieves the lowest rating and is therefore selected is 1.3.

In another aspect of the enhancement of SDG, a placement procedure is executed to arrange molecule structure diagrams closely together without overlapping. In at least some cases, the procedure is executed as a final step of SDG, after the molecule structure diagrams have been produced individually. The procedure is analytic in that the procedure does not rely on an indefinite number of iterations and is not affected by the starting positions of the components.

A specific implementation of the procedure is now described (Fig. 16), and an example as described below is illustrated in Fig. 13. A set of molecule structure diagrams and associated coordinates are acquired (step 3010). Each molecule structure diagram is represented by a conceptual box, defined by the structure diagram's smallest enclosing rectangle plus a small margin.

A "free rectangle" list is maintained that keeps track of which areas of the display area are unused (step 3020). The list is initialized to one free rectangle that occupies all of 2-D space and extends from negative infinity to positive infinity in both X and Y dimensions.

The boxes are sorted, and each is treated as follows, in order of decreasing area (step 3030). A free rectangle is selected that is closest to the center of the boxes and that is large enough to contain the instant box (step 3040). The center of a collection ("conglomeration") of boxes is defined as the average of the centers of the boxes weighted by the boxes' respective areas, or, as the center of the smallest rectangle that can enclose the boxes. The instant box is positioned flush with that corner of the free rectangle that is closest to the center of the growing collection (initially at coordinates (0,0)), and is imprinted on the free rectangle (step 3050). In imprinting, the original free rectangle is replaced by zero or more new free rectangles. New free rectangles may be created in the

leftover space, i.e., wherever the box does not occlude the original rectangle (see Fig. 12, which illustrates an example of the evolution of free rectangles and box placement). In an analogy in which a cookie cutter represents the box and dough represents the free rectangle, the dough underneath the cutter is discarded and the remaining areas of dough represent the leftover space that becomes the new free rectangles. In at least some cases, the sum of areas is not conserved, because each new free rectangle expands in both X and Y dimensions to the furthest extent possible without penetrating existing boxes. Several overlapping free rectangles may be necessary to fully span the leftover space (see Fig. 12). A free rectangle that could not contain the smallest box is not created.

In a specific implementation, overlapping free rectangles may be merged to help avoid a profusion of inconsequential free rectangles (step 3060). For example, rules may be enforced that dictate that two free rectangles should be merged such that the resulting free rectangle does not extend over any points not contained in either progenitor, provided that the percentage of area lost in the merger is less than a specified size, such as ten percent of the original area.

The conglomerate of boxes is translated so that its center is at coordinates (0,0) (step 3070). The molecule diagram coordinates are translated so that their centers coincide with their corresponding box centers (step 3080).

A practical example of the molecule arrangement procedure is illustrated in Fig. 13. Initially, a collection of molecules is presented to be positioned. Corresponding enclosing boxes are identified in an order of decreasing area (a. to d.), and are positioned one by one in the same order (i.e., a. first, d. last), to produce a space-efficient, non-overlapping arrangement as shown.

All or a portion of the procedures described above may be implemented in hardware or software, or a combination of both. In at least some cases, it is advantageous if the technique is implemented in computer programs executing on one or more programmable computers, such as a personal computer running or able to run an operating system such as UNIX, Linux, Microsoft Windows 95, 98, 2000, or NT, or MacOS, that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device such as a keyboard, and at least one output device. Program code is applied to data entered using the input device to perform the technique described above and to generate output information.

The output information is applied to one or more output devices such as a display screen of the computer.

In at least some cases, it is advantageous if each program is implemented in a high level procedural or object-oriented programming language such as Perl, C, C++, or Java to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

In at least some cases, it is advantageous if each such computer program is stored on a storage medium or device, such as ROM or optical or magnetic disc, that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other embodiments are within the scope of the following claims. For example, a non-human entity such as a computer program may serve as a source for input information such as the connection table or as a recipient of output

information such as diagrammatic data. In another example, one or more techniques based on the description herein may be applied to adapting structure diagrams for purposes other than presentation to a human user.

What is claimed is:

Claims

1. A method for use in deriving a chemical structure diagram, comprising:

identifying, from a connection table for a chemical structure, an instance of chemical structural symmetry in the chemical structure; and

expressing the instance of chemical structural symmetry in the chemical structure diagram.

2. A method for use in deriving a chemical structure diagram, comprising:

determining, from a first chemical structure diagram, a force term for increasing diagrammatic symmetry within the first chemical structure diagram; and

applying the force term in a derivation of a second chemical structure diagram from the first chemical structure diagram, the second chemical structure diagram having more diagrammatic symmetry than the first chemical structure diagram.

3. A method for use in deriving a chemical structure diagram, comprising:

determining, from a first chemical structure diagram, a parameter for use in producing the shape of an addition to the first chemical structure diagram;

producing the shape of the addition based on the parameter; and

producing a second chemical structure diagram by adding the addition to the first chemical structure diagram.

4. A method for use in deriving a chemical structure diagram, comprising:

determining a first rectangle that defines a first portion of an available layout area, the first rectangle being of a sufficient size to enclose a first chemical structure diagram;

determining a second rectangle that defines a second portion of an available layout area, the second portion being non-overlapping with the first portion, the second rectangle being of a sufficient size to enclose a second chemical structure diagram; and

positioning the first and second chemical structure diagrams within the first and second portions, respectively.

5. A system for use in deriving a chemical structure diagram, comprising:

an identifier identifying, from a connection table for a chemical structure, an instance of chemical structural symmetry in the chemical structure; and

an expressor expressing the instance of chemical structural symmetry in the chemical structure diagram.

6. A system for use in deriving a chemical structure diagram, comprising:

a determiner determining, from a first chemical structure diagram, a force term for increasing diagrammatic symmetry within the first chemical structure diagram; and

an applicator applying the force term in a derivation of a second chemical structure diagram from the first chemical structure diagram, the second chemical structure diagram having more diagrammatic symmetry than the first chemical structure diagram.

7. A system for use in deriving a chemical structure diagram, comprising:

a determiner determining, from a first chemical structure diagram, a parameter for use in producing the shape of an addition to the first chemical structure diagram; and

a producer producing the shape of the addition based on the parameter and producing a second chemical structure diagram by adding the addition to the first chemical structure diagram.

8. A system for use in deriving a chemical structure diagram, comprising:

a determiner determining a first rectangle that defines a first portion of an available layout area, the first rectangle being of a sufficient size to enclose a first chemical structure diagram, the determiner determining a second rectangle that defines a second portion of an available layout area, the second portion being non-overlapping with the first portion, the second rectangle being of a sufficient size to enclose a second chemical structure diagram; and

a positioner positioning the first and second chemical structure diagrams within the first and second portions, respectively.

9. Computer software, residing on a computer-readable storage medium, comprising a set of instructions for use in a computer system to help cause the computer system to derive a chemical structure diagram, the instructions causing the system to:

identify, from a connection table for a chemical structure, an instance of chemical structural symmetry in the chemical structure; and

express the instance of chemical structural symmetry in the chemical structure diagram.

10. Computer software, residing on a computer-readable storage medium, comprising a set of instructions for use in a computer system to help cause the computer system to derive a chemical structure diagram, the instructions causing the system to:

determine, from a first chemical structure diagram, a force term for increasing diagrammatic symmetry within the first chemical structure diagram; and

apply the force term in a derivation of a second chemical structure diagram from the first chemical structure diagram, the second chemical structure diagram having more diagrammatic symmetry than the first chemical structure diagram.

11. Computer software, residing on a computer-readable storage medium, comprising a set of instructions for use in a computer system to help cause the computer system to derive a chemical structure diagram, the instructions causing the system to:

determine, from a first chemical structure diagram, a parameter for use in producing the shape of an addition to the first chemical structure diagram;

produce the shape of the addition based on the parameter; and

produce a second chemical structure diagram by adding the addition to the first chemical structure diagram.

12. Computer software, residing on a computer-readable storage medium, comprising a set of instructions for use in a computer system to help cause the computer system to derive a chemical structure diagram, the instructions causing the system to:

determine a first rectangle that defines a first portion of an available layout area, the first rectangle being of a sufficient size to enclose a first chemical structure diagram;

determine a second rectangle that defines a second portion of an available layout area, the second portion being non-overlapping with the first portion, the second rectangle being of a sufficient size to enclose a second chemical structure diagram; and

position the first and second chemical structure diagrams within the first and second portions, respectively.

ENHANCING STRUCTURE DIAGRAM GENERATION

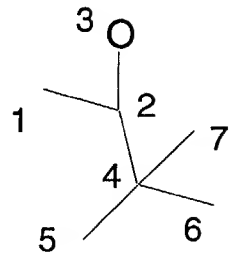
ABSTRACT OF THE DISCLOSURE

A method and a system are provided for enhancing structure diagram generation ("SDG"). In SDG, aesthetic two-dimensional ("2-D") coordinates for use in a diagrammatic representation ("diagram") of a molecule are derived from a connection table for the molecule. SDG may also improve the aesthetic qualities of a chemical structure diagram having existing coordinates, if available. SDG is enhanced by expressing the symmetry present in the molecule, by making use of symmetry in the 2-D dynamics used to lay out rings and chains, by construction of bridges using an open polygon method together with a potential function, and by an elegant approach to the relative positioning of molecules ("free rectangle method").

Atom	Elm	Bonded To
1	C	2
2	C	1,3,4
3	O	2
4	C	2,5,6,7
5	C	4
6	C	4
7	C	4

Simplified Connection Table

FIG. 1A
(PRIOR ART)



...and the molecule it represents

FIG. 1B
(PRIOR ART)

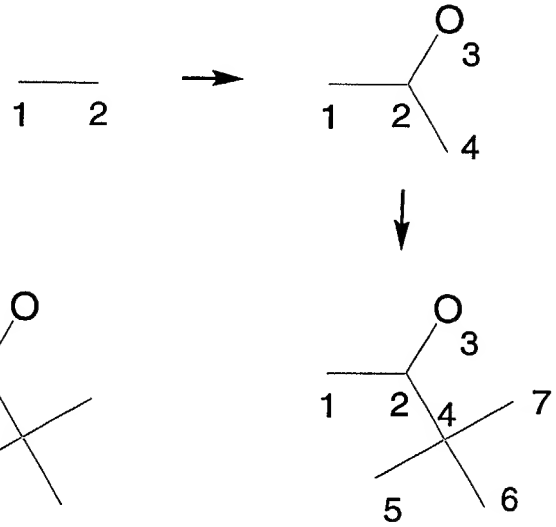
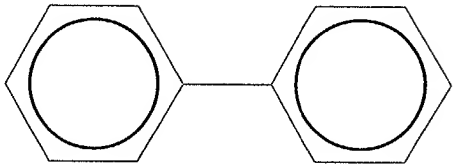
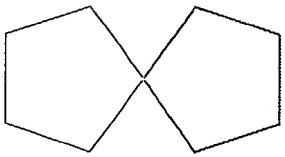


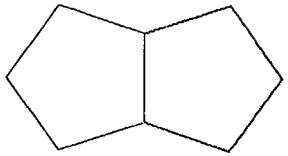
FIG. 1C



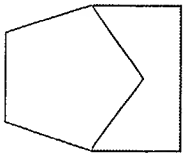
Two different ring systems are present



Spiro

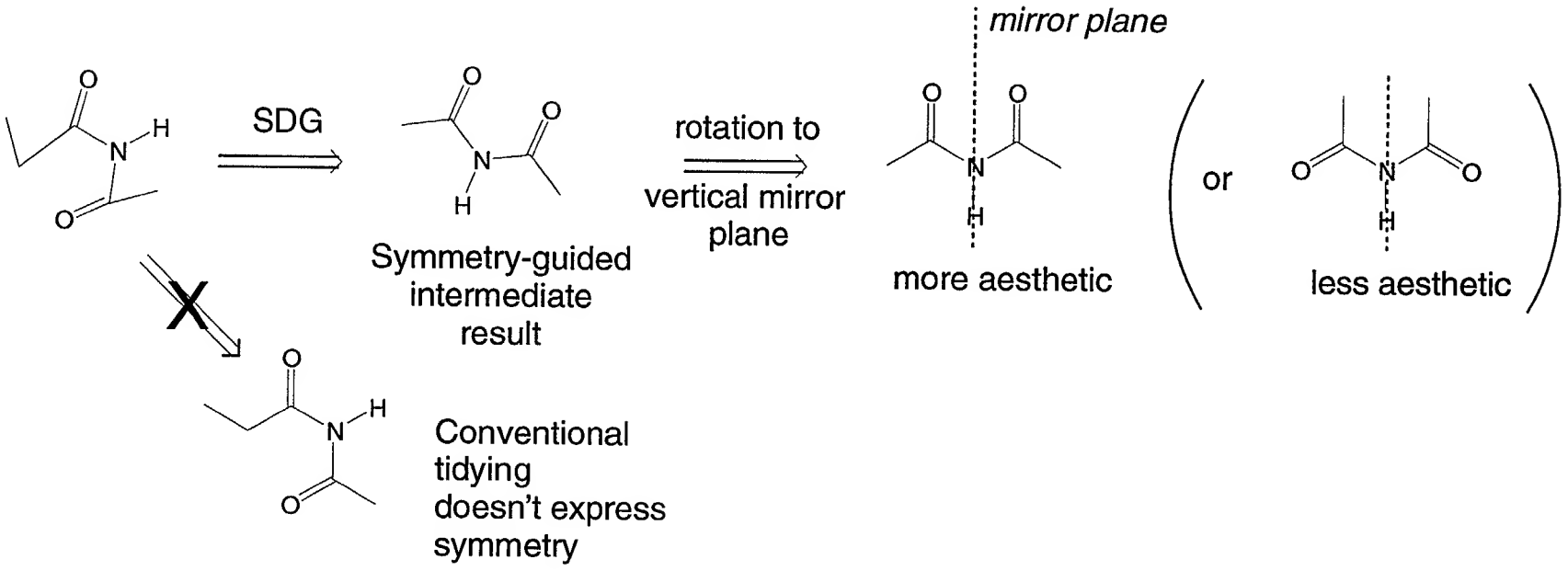


Fused

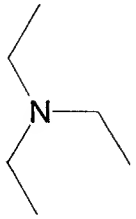


Bridged

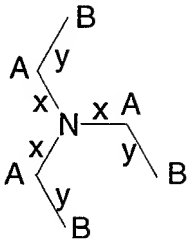
FIG. 2
(PRIOR ART)


SDG
Symmetry-guided intermediate result
rotation to vertical mirror plane
mirror plane
more aesthetic
or
less aesthetic
Conventional tidying doesn't express symmetry

F16. 3



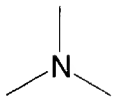
a. Given structure (Starting coordinates are irrelevant.)



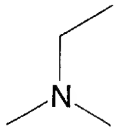
b. Perceived symmetry. Like letters indicate equivalent atoms or bonds. Symmetry is three-fold.

c. The pivot atom is taken as the first seed atom.

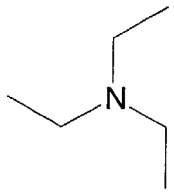
d. Place an adjacent atom.



e. Place equivalent atoms, with three-fold symmetry

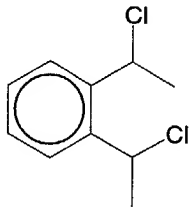


f. Place next atom. (Direction is arbitrary.)

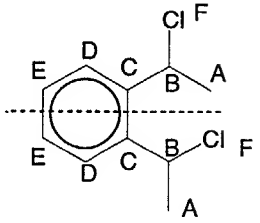


g. Place equivalent atoms, with three-fold symmetry.

FIG. 4

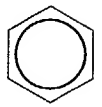


a. Given structure (Starting coordinates are irrelevant.)

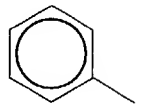


b. Perceived symmetry. Like letters indicate equivalent atoms. Symmetry is reflection.

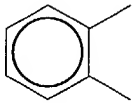
c. Deposit first atom.



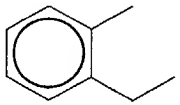
d. Because it is cyclic, we deposit the whole ring as one unit.



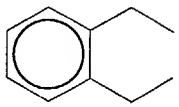
e. Place next atom.



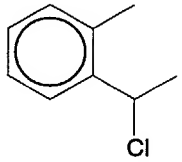
f. Place equivalent atom, with reflectional symmetry



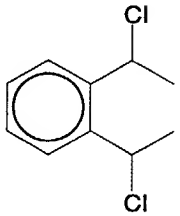
g. Place next atom. (Direction is arbitrary.)



h. Place equivalent atom, with reflectional symmetry.



i. Place next atom.



j. Place equivalent atom, with reflectional symmetry.

FIG. 5

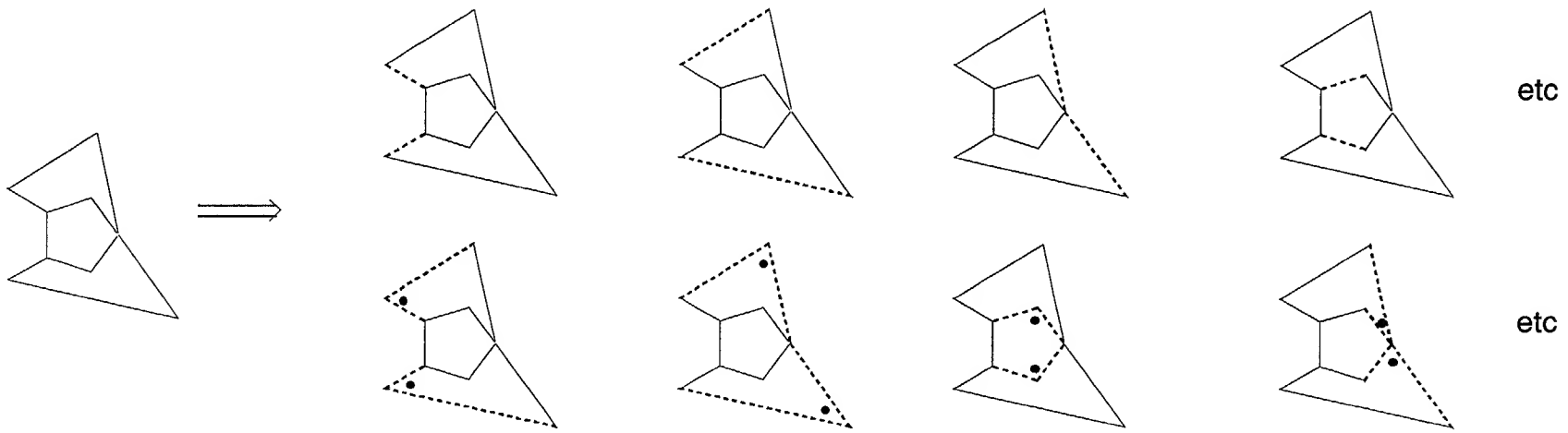


FIG. 6

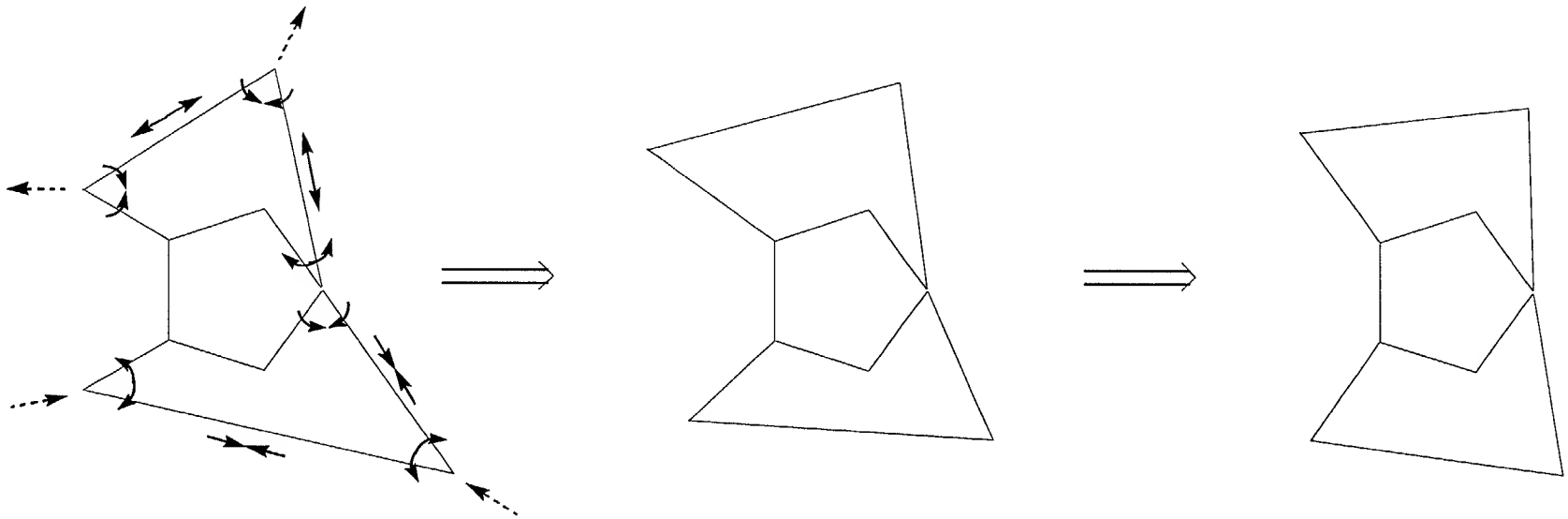
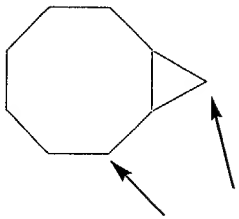
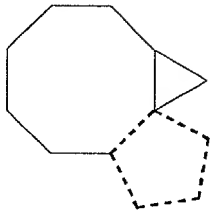


FIG. 7

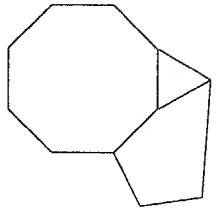


Goal: Create a five membered ring attached at the points shown.

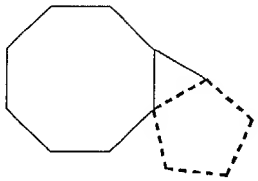
Attach regular polygon



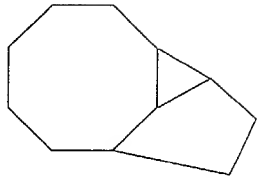
Final Result



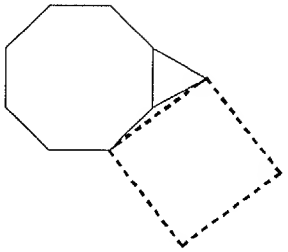
Attach regular polygon



Final Result



Use outer atoms



Final Result

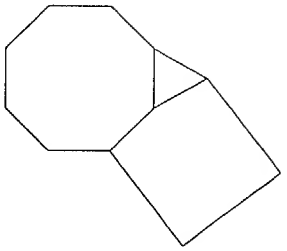


FIG. 8

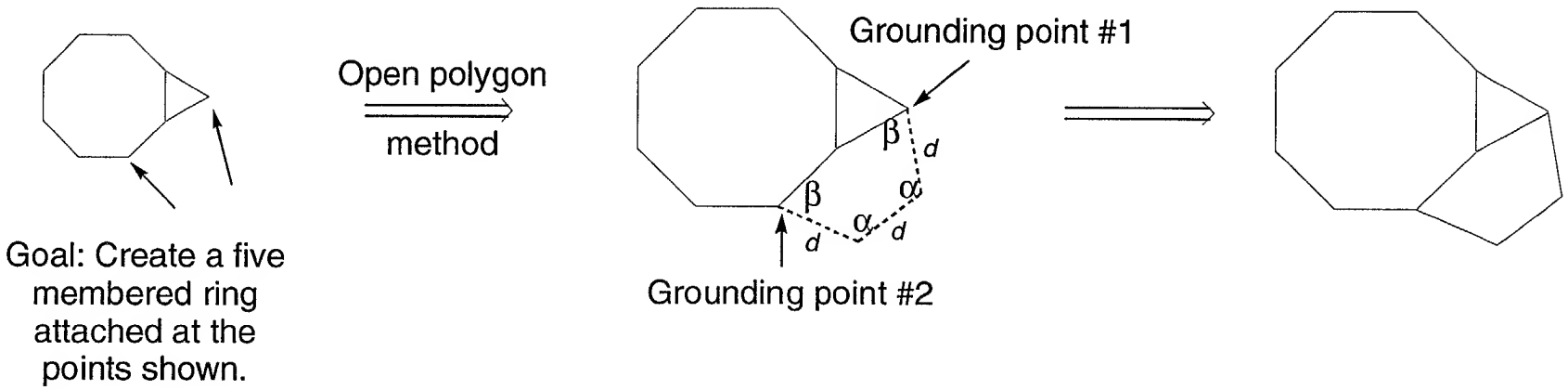

Goal: Create a five membered ring attached at the points shown.
Open polygon
method
Grounding point #1
β
d
β
α
α
d
d
Grounding point #2

FIG. 9

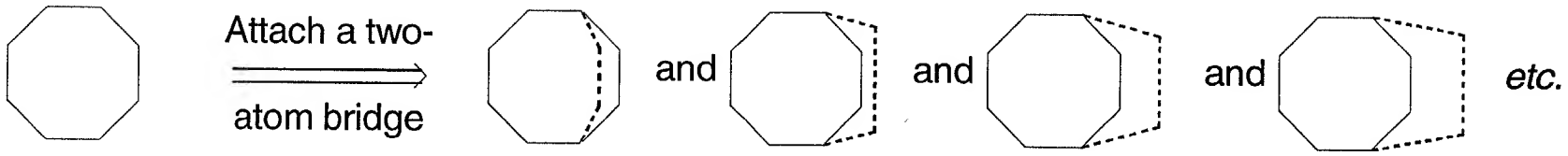
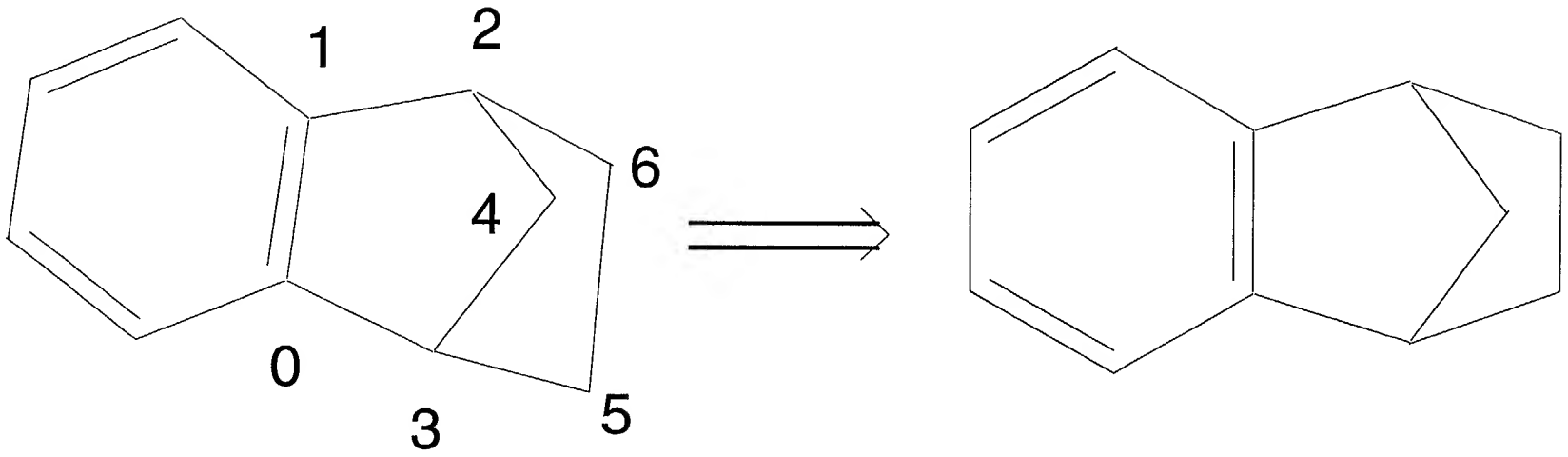
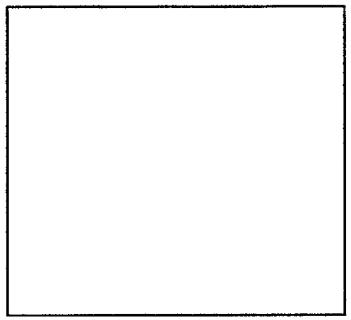


FIG. 10

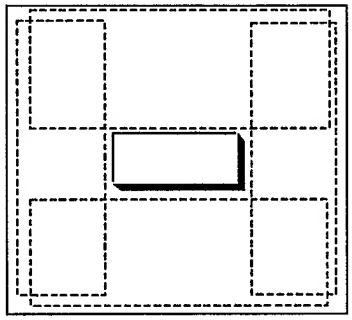



```
| ------------------------------------------------------------------Enter RD_AttachPeeledBridge [3]
| Attaching peeled bridge at atoms 2 (CW) and 3 (CCW)
| Rating for bd len scale 1.0 is  198.294 (= congest[ 178.294] + BdLen[0.00] + bdAng[ 20 (bdAng=125)])
| Rating for bd len scale 0.5 is  313.185 (= congest[  45.185] + BdLen[40.00] + bdAng[228 (bdAng=177)])
| Rating for bd len scale 0.7 is  291.643 (= congest[ 179.643] + BdLen[24.00] + bdAng[ 88 (bdAng=142)])
| Rating for bd len scale 0.9 is  242.107 (= congest[ 178.107] + BdLen[8.00] + bdAng[ 56 (bdAng=134)])
| Rating for bd len scale 1.1 is   93.917 (= congest[  85.917] + BdLen[8.00] + bdAng[  0 (bdAng=119)])
| Rating for bd len scale 1.3 is   56.154 (= congest[  32.154] + BdLen[24.00] + bdAng[  0 (bdAng=109)])
| Rating for bd len scale 1.5 is   61.044 (= congest[  21.044] + BdLen[40.00] + bdAng[  0 (bdAng=103)])
| Rating for bd len scale 1.7 is   72.576 (= congest[  16.576] + BdLen[56.00] + bdAng[  0 (bdAng=98)])
| Rating for bd len scale 1.9 is   86.400 (= congest[  14.400] + BdLen[72.00] + bdAng[  0 (bdAng=94)])
| Ring 1: Best bridge scale factor = 1.30 (rating =   56.154)
| Irregular polygon. (numAtsToDraw=4; RNGSIZ=5; aOuter_CW=2; _CCW=3)
| ------------------------------------------------------------------ Exit RD_AttachPeeledBridge
```

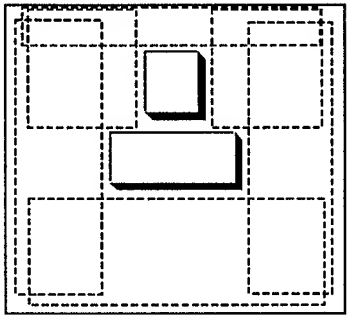
FIG. 11



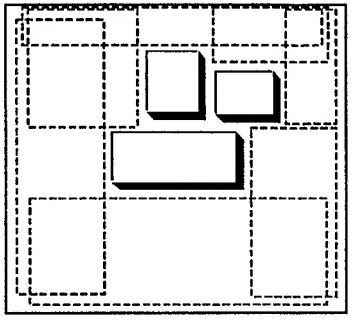
a. Initial free rectangle



b. After imprinting the first box, there are four free rectangles.

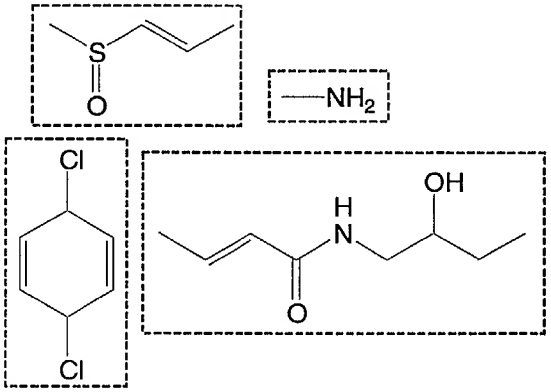


c. After imprinting the second box, there are seven free rectangles. (Translation step not included for clarity.)

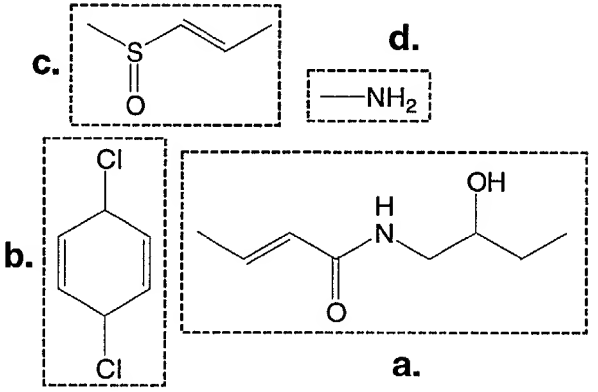


d. After imprinting the third box, there are eight free rectangles. (Translation step not included for clarity.)

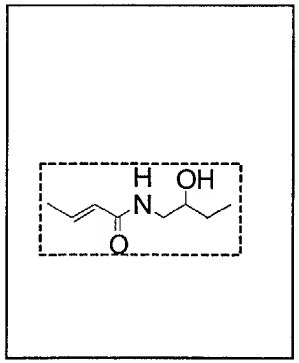
FIG. 12



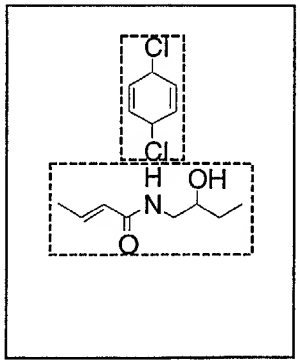
1. A collection of molecules to be positioned, with their enclosing boxes.



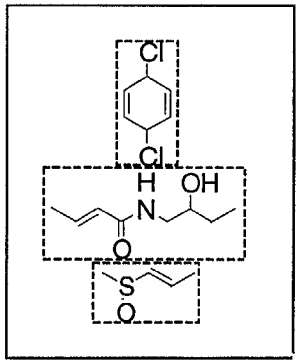
2. Boxes sorted by decreasing area.



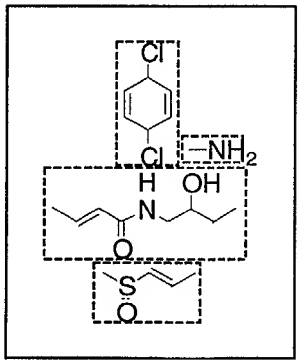
3. After placing the largest box.



4. After placing second box.



5. After placing third box.



6. After placing fourth box.

FIG. 13

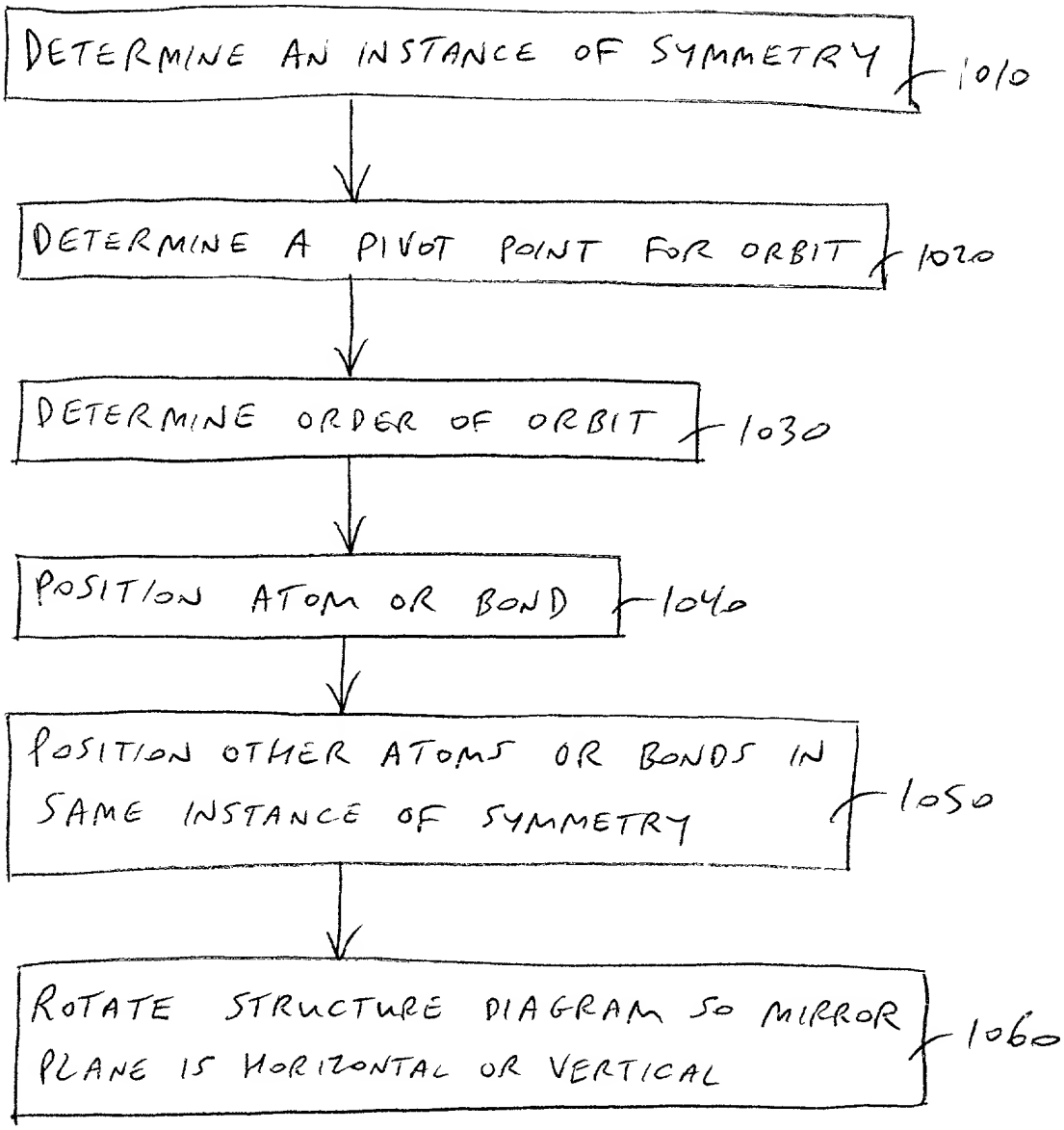

DETERMINE AN INSTANCE OF SYMMETRY
1010
DETERMINE A PIVOT POINT FOR ORBIT
1020
DETERMINE ORDER OF ORBIT
1030
POSITION ATOM OR BOND
1040
POSITION OTHER ATOMS OR BONDS IN SAME INSTANCE OF SYMMETRY
1050
ROTATE STRUCTURE DIAGRAM SO MIRROR PLANE IS HORIZONTAL OR VERTICAL
1060

FIG. 14

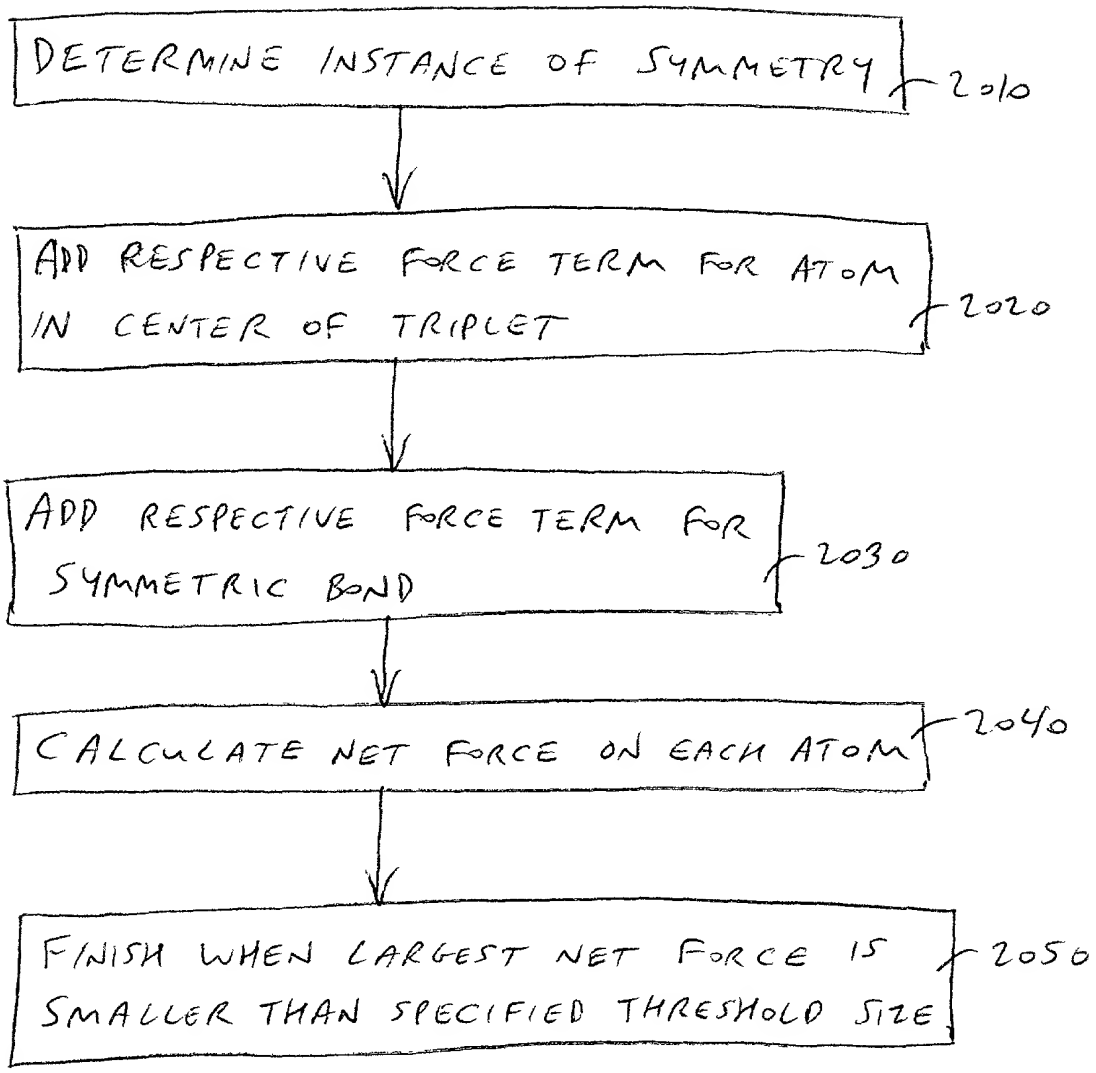

DETERMINE INSTANCE OF SYMMETRY
2010
ADD RESPECTIVE FORCE TERM FOR ATOM IN CENTER OF TRIPLET
2020
ADD RESPECTIVE FORCE TERM FOR SYMMETRIC BOND
2030
CALCULATE NET FORCE ON EACH ATOM
2040
FINISH WHEN LARGEST NET FORCE IS SMALLER THAN SPECIFIED THRESHOLD SIZE
2050

FIG. 15

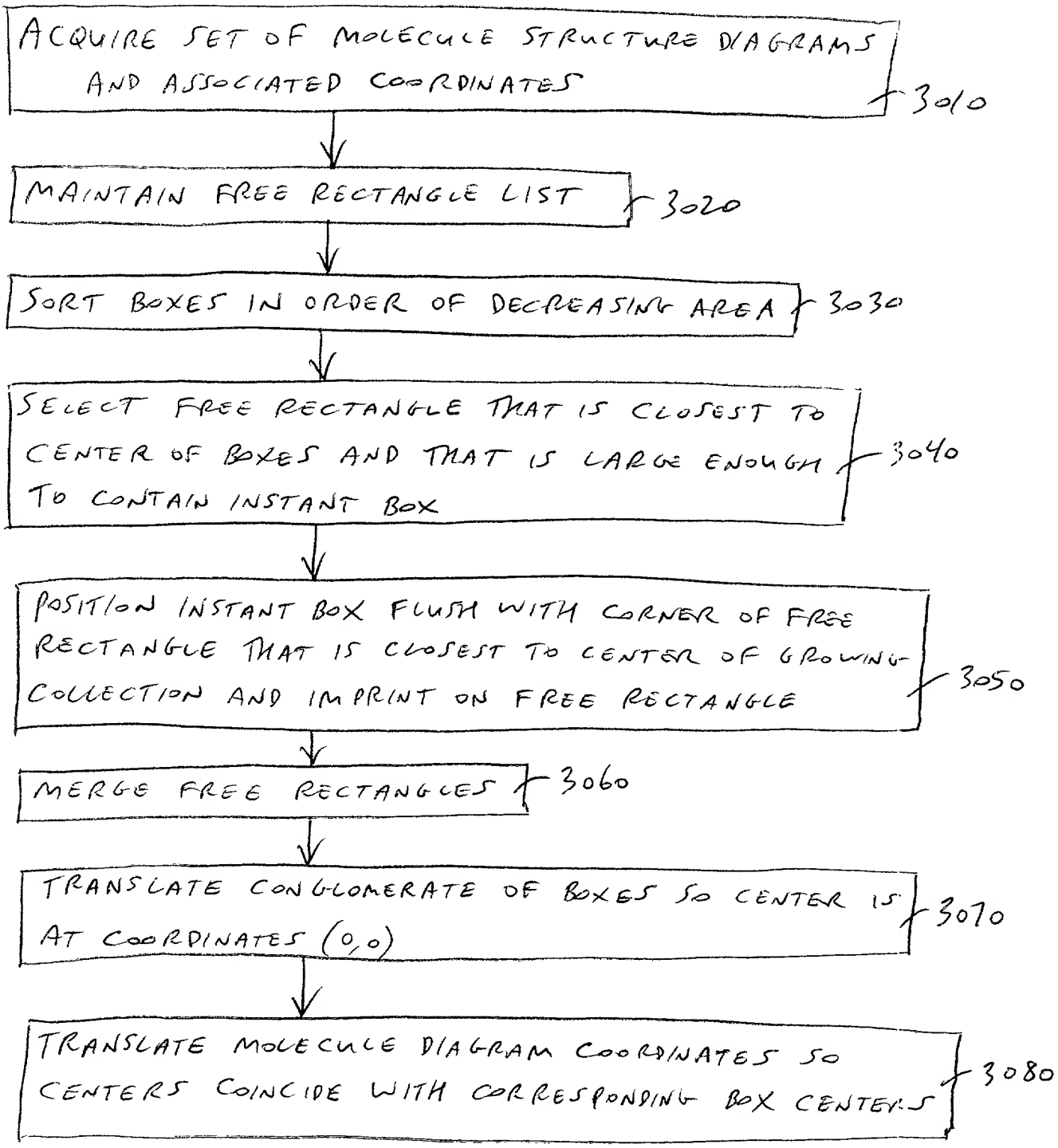

ACQUIRE SET OF MOLECULE STRUCTURE DIAGRAMS AND ASSOCIATED COORDINATES
3010
MAINTAIN FREE RECTANGLE LIST
3020
SORT BOXES IN ORDER OF DECREASING AREA
3030
SELECT FREE RECTANGLE THAT IS CLOSEST TO CENTER OF BOXES AND THAT IS LARGE ENOUGH TO CONTAIN INSTANT BOX
3040
POSITION INSTANT BOX FLUSH WITH CORNER OF FREE RECTANGLE THAT IS CLOSEST TO CENTER OF GROWING COLLECTION AND IMPRINT ON FREE RECTANGLE
3050
MERGE FREE RECTANGLES
3060
TRANSLATE CONGLOMERATE OF BOXES SO CENTER IS AT COORDINATES (0,0)
3070
TRANSLATE MOLECULE DIAGRAM COORDINATES SO CENTERS COINCIDE WITH CORRESPONDING BOX CENTERS
3080

FIG. 16

DECLARATION AND POWER OF ATTORNEY
(Attorney Docket No. 103544.127)

As a below-named inventor, I hereby declare that:

My residence, post office address and citizenship is as stated below next to my name.

I believe that I am the original, first and sole inventor (if only one name is listed be.ow) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

ENHANCING STRUCTURE DIAGRAM GENERATION

the specification of which (check only one):

[X] is attached hereto.

[ ] was filed as United States Patent Application
Serial No._____
and was amended
on _____
(if applicable)

[ ] was filed as PCT Patent Application
Serial No. _____
on _____
and was amended under PCT Article 19
on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of the claims of this application in accordance with Title 37, Code of Federal Regulations, Sections 1.56(a) and 1.56(b).

I hereby claim foreign priority benefits under Title 35, United States Code, §119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

EXPRESS MAIL LABEL NO. EM259723534US
DATE OF DEPOSIT February 11, 2000

PRIOR FOREIGN/PCT APPLICATION(S) AND ANY PRIORITY CLAIMS UNDER 35 U.S.C. §119(a)-(d) or 365(b):

COUNTRY (if PCT indicate PCT)	APPLICATION NUMBER	DATE OF FILING	PRIORITY CLAIMED UNDER 35 U.S.C. §119(a)-(b) or 365(b) (YES/NO)

I hereby claim the benefit under 35 U.S.C. §119(e) of any United States provisional patent application(s) listed below:

APPLICATION NUMBER	DATE OF FILING	STATUS: (PENDING OR ABANDONED)
60/119,654	February 11, 1999	PENDING

I hereby claim the benefit under Title 35, United States Code, § 120 or 365(c) of any United States application(s) or PCT international application(s) designating the United States of America that is/are listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in that/those prior application(s) in the manner provided by the first paragraph of Title 35, United States Code, § 112. I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56 which occurred between the filing date of the prior applications and the national or PCT international filing date of this application:

PRIOR U.S. APPLICATION OR PCT INTERNATIONAL APPLICATION(S) DESIGNATING THE U.S. FOR BENEFIT UNDER 35 U.S.C. § 120 or 365(c):

APPLICATION NUMBER	DATE OF FILING (day, month, year)	STATUS: (PATENTED, PENDING OR ABANDONED)

POWER OF ATTORNEY: As named inventors, we hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith

Michael J. Bevilacqua	Reg. No. 31,091
James B. Lampert	Reg. No. 24,564
Wayne M. Kennard	Reg. No. 30,271
Hollie L. Baker	Reg. No. 31,321
Wayne A. Keown	Reg. No. 33,923
Donald R. Steinberg	Reg. No. 37,241
Michael A. Diener	Reg. No. 37,122
Richard A. Goldenberg	Reg. No. 38,895

Peter M. Dichiara	Reg. No. 38,005
Ann-Louise Kerner	Reg. No. 33,523
Colleen Superko	Reg. No. 39,850
Gretchen Rice	Reg. No. 37,429
Keum J. Park	Reg. No. 42,059
Jason A. Reyes	Reg. No. 41,513
Janice M. Klunder	Reg. No. 41,121
Henry N. Wixon	Reg. No. 32,073
Barbara A. Barakat	Reg. No. 32,190
Nancy Chiu	Reg. No. 43,545
Rajesh Vallabh	Reg. No. 35,761
Ayla A. Lari	Reg. No. 43,739

the mailing address and telephone number of each of whom is HALE AND DORR LLP, 60 State Street, Boston, Massachusetts 02109 and (617) 526-6000, with full power of substitution and revocation to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith.

Send Correspondence To

Jason A. Reyes
Hale and Dorr LLP
60 State Street
Boston, MA 02109

Direct Telephone Calls To

Jason A. Reyes
(617) 526-6010

Wherefore I petition that letters patent be granted to me for the invention or discovery described and claimed in the attached specification and claims, and hereby subscribe my name to said specification and claims and to the foregoing declaration, power of attorney, and this petition.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole inventor: Harold E. Helson

Inventor's signature _____ Date _____

Country of Citizenship: USA

Residence: 69 Bartlett Avenue, Arlington, MA 02476

Post Office Address: _____

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Harold E. Helson

Serial No.: To Be Assigned

Filed: Herewith (This application claims the benefit of U.S. Provisional Application Serial No. 60/119,654 entitled STRUCTURE DIAGRAM GENERATION, filed on February 11, 1999.)

Title: ENHANCING STRUCTURE DIAGRAM GENERATION

Box Patent Application
Assistant Commissioner for Patents
Washington, DC 20231

COVER SHEET FOR SOURCE CODE APPENDIX

Dear Sir:

Enclosed for filing in the above-referenced patent application is the following document:

1. Source Code Appendix, 30 pages.

The following is the inventor's residence:
69 Bartlett Avenue, Arlington, MA 02476.

Respectfully submitted,

Dated: February 11, 2000

Jason A. Reyes
Registration No. 41,513
Attorney for Applicant

Hale and Dorr LLP
60 State Street
Boston, MA 02109
Tel.: (617) 526-6010
Fax: (617) 526-5000

Attorney Docket No. 103544.127

EXPRESS MAIL LABEL NO. EM259723534US
DATE OF DEPOSIT February 11, 2000

/*

File: :sdg:sdg_ringDesign.cpp

Contains: Computes coordinates of ring systems given a ring strategy.

$Header: /ChemDraw/Src/sdg/sdg_ringDesign.cpp 41 12/23/99 6:32p Jsb $

/*

+----------+

HEH 01/20/99 CDBR-6450: Changed m_asRingAtomsPlaced_frg into m_asRingAtomsPlaced_RDU. |

HEH 01/15/99 RD_AttachPeeledBridge(): When choosing bridge position, penalize linear bds.|

HEH 01/14/99 RD_AttachPeeledBridge(): Include bds adjacent to border ats in congest.calc.|

HEH 01/07/99 ComputeCongestion(): Replace ad hoc in-place code with calling Red_Potent().|

HEH 01/05/99 Lengthen or contract bridge to avoid overlap with already-laid down parts. |

HEH 12/21/98 CFBR-4853: RD_AttachPeeledBridge(): Draw bridge on less congested side. |

HEH 12/13/98 Added DYNAMIC ring strategy.

|

HEH 09/02/97 Added RA_AreAtomsOrBondsContiguousAboutRing(). Moved RingTransit to CC. |

HEH 07/29/96 RD_DesignRing(): Clear CFS_definedV of spiro atoms at end of RDU.

|

HEH 07/29/96 RD_MakeSimpleCore(): Add RINGS_REST_ON_FLAT_EDGE

|

HEH 07/19/96 New class RingTransit:: supplants TraverseRing(). ChasePolygon() becomes |
obsolete.

|

HEH 07/19/96 Ring drawing order was determined in ring design; now in ring strategy. |

HEH 07/19/96 New fn RD_MakeSimpleCore(); MakeSimpleRingSystem() is obsolete.

|

```
HEH  07/19/96   Handles bridges.  New fn RD_AttachPeeledBridge().
|
HEH  07/19/96   Renamed AttachRing() to RD_AttachSimpleRing().
|
HEH  07/19/96   New fn RD_AttachThing() places a ring's atoms and calculates CFS's given      |
                    a vector of coordinates.  Extracted from old AttachRing() so as to
treat  |
                    the commonality between peeled simple and peeled bridge.
|
+-------------------------------------------------------------------------------------+
*/


//-------------------------------------------------------------------------------
class RD_BridgeCongestionEnvironment
{
public:
                                RD_BridgeCongestionEnvironment (SDG &c, int
ringNum, ccRingTransit &ringTransit, ATOMNO aOuter_CW, ATOMNO aOuter_CCW, SREF asUndrawnAtoms, SREF
asDrawnAtoms, sdgFloat bdLen);
    sdgFloat                ComputeCongestion();
    bool                    IsCongested()  const  { return m_congestion > 20.; }

    int                       m_ringNum;
    ccRingTransit&          m_ringTransit;
    ccPoint2D                 m_P1, m_P2;
    vector<ccPoint2D>       m_coords;
    int                       m_numAtsToDraw;
    sdgFloat                m_bdLen;
```

```
    const ccSet&        m_asUndrawnAtoms,
                            m_asDrawnAtoms;
    ATOMNO                  m_aOuter_CW, m_aOuter_CCW;
    sdgFloat                m_congestion;  // squirreled copy of value found in
ComputeCongestion().
    vector<ccPoint2D>       m_trialCoords;
    sdgFloat                m_polyPhi;
    SDG&                    C;
};
//-------------------------------------------------------------------------------------
RD_BridgeCongestionEnvironment::RD_BridgeCongestionEnvironment (SDG &c, int ringNum, ccRingTransit
&ringTransit, ATOMNO aOuter_CW, ATOMNO aOuter_CCW, SREF asUndrawnAtoms, SREF asDrawnAtoms, sdgFloat
bdLen)
    :   C                       (c)
    ,   m_ringNum               (ringNum)
    ,   m_ringTransit       (ringTransit)
    ,   m_numAtsToDraw      (asUndrawnAtoms.NMems() + 2)   // includes the two border
atoms
    ,   m_coords                (asUndrawnAtoms.NMems() + 2)   // ditto
    ,   m_bdLen                 (bdLen)
    ,   m_asUndrawnAtoms    (asUndrawnAtoms)
    ,   m_asDrawnAtoms      (asDrawnAtoms)
    ,   m_aOuter_CW             (aOuter_CW)
    ,   m_aOuter_CCW        (aOuter_CCW)
    ,   m_trialCoords       ((asUndrawnAtoms | asDrawnAtoms).Last() + 1)
{
    ASSERT (m_numAtsToDraw == m_asUndrawnAtoms.NMems() + 2);      // m_numAtsToDraw includes
the two drawn rooted atoms
    m_P1 = C.GetVXY (m_aOuter_CW);
```

```
    m_P2 = C.GetVXY (m_aOuter_CCW);
    const bool     LVal = C.RD_OpenPolygon (m_P1, m_P2, m_numAtsToDraw, m_bdLen,
kccCounterClockwise, m_coords, &m_polyPhi);

    m_ringTransit.MoveTo (m_aOuter_CCW);
    for (int x = 2;  x < m_numAtsToDraw;  x++)      // skip the two rooted atoms
    {
        m_ringTransit.Advance();
        m_trialCoords [m_ringTransit.Curr()] = m_coords [x];
        ASSERT (m_asUndrawnAtoms.IsMem (m_ringTransit.Curr()));
    }
    ATOMNO  a;
    LOOP_SET (m_asDrawnAtoms, a)    // avoid executing this loop many times by moving
m_trialCoords outside of this object
        m_trialCoords [a] = C.GetVXY (a);
}
//-------------------------------------------------------------------------------
sdgFloat RD_BridgeCongestionEnvironment::ComputeCongestion()
{
    const ccSet    asTwoBorderAtoms = ccMakeSet (m_aOuter_CW, m_aOuter_CCW);
    const ccSet    asInterestingDrawnAtoms   = m_asDrawnAtoms - asTwoBorderAtoms;
    const ccSet bsInterestingDrawnBonds   = C.CT.JoiningBonds (m_asDrawnAtoms),
                bsInterestingUndrawnBonds = C.CT.JoiningBonds (m_asUndrawnAtoms |
asTwoBorderAtoms);

    m_congestion = C.Potential_BB ( bsInterestingUndrawnBonds,     m_trialCoords,
                                    bsInterestingDrawnBonds,
m_trialCoords, 3.0);
    CDBG2 ( sdgOut ("ComputeCongestion: total congestion = %8.3lf\n") << m_congestion; )
```

```
    return m_congestion;
}
//---------------------------------------------------------------------------


/*
+=========================================================================================+
|
|
| LinearAnglePenalty    Calculate a penalty for near-linear bonds.
|
|
|
+-----------------------------------------------------------------------------+
|  Penned by H.Helson, 1/15/99.
|
+=========================================================================================+
*/
static int  LinearAnglePenalty (const sdgFloat &ang, int threshhold_deg = 120)
{
    const int     interiorBondAngle_deg = 180 - RtoD (ang);
    const int     badAnglePenalty = 4 * max (0, interiorBondAngle_deg - threshhold_deg);
    return badAnglePenalty;
}

/*
+=========================================================================================+
|
|
```

```
| RD_AttachPeeledBridge
|
|
|
+-------------------------------------------------------------------------------+
| [R-] rngNo              The ring to be merged in.
|
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|     kAtten is a crude patch to prevent [m.n.n] systems from receiving overlapping bridges. |
|     asUndrawnAtoms includes frozen atoms.
|
|
|
|  Penned by H.Helson, 7/12/96.
|
+===============================================================================================+
*/
void CClean::RD_AttachPeeledBridge (int rngNo)
{
      ENTER1 ("RD_AttachPeeledBridge");
      const ccCW_Sense      sense = kccCounterClockwise;
      const sdgFloat        kAtten = 0.75;
      const ccSet               asUndrawnAtoms = RI.GetAtoms (rngNo) -
m_asRingAtomsPlaced_RDU;
      const ccSet               asBorderAtoms = CT.Alpha_AA (asUndrawnAtoms) &
m_asRingAtomsPlaced_RDU;
      ASSERT (asBorderAtoms.NMems() == 2);
      if (asBorderAtoms.NMems() != 2) // avoid possible memory corruption by skipping out now.
            throw sdgException (sdgException::kAvoidMemCorruption, "RD_AttachPeeledBridge");
```

```
    ATOMNO  aBorder_1, aBorder_2;
    asBorderAtoms.Bits12 (aBorder_1, aBorder_2);
    ccRingTransit   hobbit (M, rngNo, cckAtom, sense);
    hobbit.MoveTo (aBorder_1);

    if (m_asRingAtomsPlaced_RDU.IsMem (hobbit.Prev()))
        Swap (aBorder_1, aBorder_2);

#ifdef _DEBUG
    hobbit.MoveTo (aBorder_1);
    ASSERT (!m_asRingAtomsPlaced_RDU.IsMem (hobbit.Prev()) &&  m_asRingAtomsPlaced_RDU.IsMem
(hobbit.Next()));
    hobbit.MoveTo (aBorder_2);
    ASSERT ( m_asRingAtomsPlaced_RDU.IsMem (hobbit.Prev()) && !m_asRingAtomsPlaced_RDU.IsMem
(hobbit.Next()));
#endif

    // Invert which side the bridge is drawn on, depending on relative lengths of the current
    // and the drawn bridge.  (CDBR-4853)
    // For example bicyclo[10.5.1]alkane: Optimally, first the seventeen-membered ring is
    // drawn perfect-polygonally, ff. by the one-membered leg.  But if a thirteen-ring is
    // first drawn, the remaining 5-leg should be drawn on the outside, not the inside.
    if (hobbit.Distance (aBorder_1, aBorder_2) - 1 < (RI.Size (rngNo) - 2) / 2)     // "-2" to
discount bridgeheads
    {
        Swap (aBorder_1, aBorder_2);
        hobbit.ReverseSense();
    }
```

```
    CDBG ( sdgOut ("Attaching peeled bridge at atoms %d (CW) and %d (CCW)\n") << aBorder_1 <<
aBorder_2; )
    const int           numAtsToDraw = asUndrawnAtoms.NMems() + 2;     // include the
border atoms
    const sdgFloat  bdLen = m_bndLen_F * kAtten;

    // If the newly placed bridge overlaps part of the ring system already laid down, try
    // increasing or decreasing its bond lengths.
    const ccSet     &asDrawnAtoms = m_asRingAtomsPlaced_RDU;
    DBG ( const char formatMsg[] = "Rating for bd len scale %3.1lf is %8.3lf (= congest[%8.3lf]
+ BdLen[%3.2lf] + bdAng[%3d (bdAng=%d)])\n"; )

    RD_BridgeCongestionEnvironment  bce_normal (*this, rngNo, hobbit, aBorder_1, aBorder_2,
asUndrawnAtoms, asDrawnAtoms, bdLen);
    const sdgFloat  congest_normal = bce_normal.ComputeCongestion();
    const int           badAnglePenalty = LinearAnglePenalty (bce_normal.m_polyPhi);   //
penalize near-linear bonds
    sdgFloat            rating_best = congest_normal + badAnglePenalty,
                          scale_best  = 1.0;
    CDBG0 ( sdgOut (formatMsg) << scale_best << rating_best << congest_normal << 0. <<
badAnglePenalty << 180-RtoD (bce_normal.m_polyPhi); )
    if (bce_normal.IsCongested())
    {
        for (sdgFloat scale = 0.5;  scale < 2.0;  scale += 0.2) // misses scale=1.0, which
was covered above
        {
            RD_BridgeCongestionEnvironment  bce_trial (*this, rngNo, hobbit, aBorder_1,
aBorder_2, asUndrawnAtoms, asDrawnAtoms, bdLen * scale);
            const sdgFloat  congest_trial = bce_trial.ComputeCongestion();
```

```
                const sdgFloat  nonStandardBondLengthPenalty = 80 * abs (scale - 1.0);  //
arbitrary penalty for not using std bond length
                const int               badAnglePenalty = LinearAnglePenalty
(bce_trial.m_polyPhi);
                const sdgFloat  rating_trial = congest_trial + nonStandardBondLengthPenalty
+ badAnglePenalty;
                CDBG0 ( sdgOut (formatMsg) << scale << rating_trial << congest_trial <<
nonStandardBondLengthPenalty
                                        << badAnglePenalty << 180-RtoD
(bce_trial.m_polyPhi); )
                if (rating_trial < rating_best)
                {
                        rating_best = rating_trial;
                        scale_best  = scale;
                }
            }
        }
        CDBG ( sdgOut ("Ring%2d: Best bridge scale factor = %3.2lf (rating = %8.3lf)\n") << rngNo
<< scale_best << rating_best; )

        RD_AttachThing (rngNo, aBorder_1, aBorder_2, hobbit, aBorder_1, aBorder_2, numAtsToDraw,
bdLen * scale_best);

}



/*
```

File: :sdg:sdg_repo.cpp

Contains: Repositions fragments after they are designed de novo.

$Header: /ChemDraw/Src/sdg/sdg_repo.cpp 28 12/23/99 6:32p Jsb $

```
+-------------------------------------------------------------------------------+
HEH  07/12/99   [Pending] Reposition_Analytic(): Reposition all fragments, not just redrawable
ones.  |
HEH  05/09/99   Reposition(): now available on request by the kReposition opflag.
|
HEH  03/15/99   Reposition_Analytic(): Fix mem err caused by incorrect dimensioning.          |
HEH  01/22/99   Reposition_Analytic(): Avoid divide-by-zero when m_stdBondLen_W provided zero.|
SDR  01/05/99   Avoid conflict with Mac toolbox "topLeft" macro
|
HEH  12/05/98   Reposition_Analytic(): When no bonds, use standard bond length.
|
HEH  11/30/98   CDBR-3905: Added Reposition_Analytic() using simple "dynamic grid" alg.       |
HEH  11/30/98   CAMEO's FreeRect algorithm adapted for use here (in C++) as sdgFreeRect::     |
HEH  11/30/98   File created.
|
+-------------------------------------------------------------------------------+
*/

#include "sdg.h"
#include <limits>
using namespace std;
```

SET_OUTPUT_LEVEL2 (0, &SDG::sdgMasterOutputLevel)

```
/*
+=============================================================================================+
| sdgFreeRect  This class is used to locate a fre rectangle in a given target area and a      |
|                    list of rectangles that are off limits.  Represents a given
rectangulaar   |
|                    area (A) as a patchwork of "free" (available) rectangles.  These
rectangles    |
|                    do overlap and all reside within A.
|
+=============================================================================================+
*/
class sdgFreeRect      // Method adapted from this author's approach in CAMEO's DynaJump routine.
{
public:
     // METHODS
                         sdgFreeRect (ccRect targetRectangle = ccRect (0,0,0,0),
bool mergeAdjacentRects = true);
     bool                RegisterOccupiedRectangle (const ccRect &occRect);

     typedef list<ccRect>::iterator  FrIter;

private:
     void                AddFR (FrIter pInsertBefore, long left, long top, long right, long
bottom);
     void                DelFR (FrIter pFR);
DBG(void                     Dump (FrIter pFR); )
```

```
DBG(void                Dump(); )
DBG(void*               FrAddress (const FrIter pIt); )

    // DATA
public:
    list<ccRect>   m_freeRects;

private:
    ccRect          m_targetRect;
    unsigned long   m_numPasses;   // for debugging only
    bool            m_merge;
    static bool     m_skipSmall;
};
bool   sdgFreeRect::m_skipSmall = true;// don't waste time with very small regions


/*
+=============================================================================+
| sdgFreeRect ctor
|
+=============================================================================+
*/
sdgFreeRect::sdgFreeRect (ccRect targetRectangle, bool mergeAdjacentRects)
    :   m_targetRect   (targetRectangle)
    ,   m_merge        (mergeAdjacentRects)
    ,   m_numPasses    (0)
{
    if (m_targetRect.IsRectEmpty())
    {
```

```
        const long      kMin = -numeric_limits<long>::min() / 4,
                        kMax =  numeric_limits<long>::max() / 4;
        m_targetRect.Set (kMin, kMin, kMax, kMax);
        m_merge = false;
    }
    AddFR (m_freeRects.end(), m_targetRect.left, m_targetRect.top, m_targetRect.right,
m_targetRect.bottom);
}

/*
+=============================================================================================+
| AddFR       Create a new FR.  The new FR is inserted just before pInsertBefore.
|
+=============================================================================================+
*/
void sdgFreeRect::AddFR (FrIter pInsertBefore, long left, long top, long right, long bottom)
{
    ENTER1 ("AddFR");
    const double    kPercentage = .05;          // percentage of dimension.

    // Check if new FR is entirely within an already existing one.
    // This should not occur if the algorithm is functioning properly.
    for (FrIter check = m_freeRects.begin();  check != m_freeRects.end();  check++)
    {
        if (check == pInsertBefore)     // pInsertBefore IS a superset; however it may
perish shortly
            continue;
        bool    inside = false;
        if (check->left <= left  &&  check->right  >= right)
```

```
            if (check->top <= top && check->bottom >= bottom)
                inside = true;
        if (check->left >= left  && check->right <= right)
            if (check->top >= top && check->bottom <= bottom)
                inside = true;
        if (inside)
        {
            CDBG1 ( sdgOut ("Proposed FR (%3ld..%3ld)(%3ld..%3ld) is inside preexisting
FR %lX; pInsertBefore=%lX\n") <<
                                    left << right << top << bottom << FrAddress
(check) << FrAddress (pInsertBefore); )
            return;
        }

        // Check if new FR borders a preexisting one; if so, merge.
        if (!m_merge)
            continue;
        const long    width  =  right - left;
        const long    height = bottom - top;
        if (abs (check->left  - left)  < (int)(kPercentage * (float)width)  &&
            abs (check->right - right) < (int)(kPercentage * (float)width))
        // Horizontal dimension aligns; check for overlap in V
        {
            if (check->bottom >= bottom)
            {
                if (check->top <= bottom)      // overlap in Y?: new FR is above &
overlapping
                {
#ifdef _DEBUG
```

```
                        CDBG2 ( sdgOut << "Merging two FR's #1" << endl; )
                        CDBG2 ( sdgOut ("Merging new (%ld..%ld, %ld..%ld) with old
%lX; Before:\n") << left << right << top << bottom << FrAddress (check); )
                        CDBG2 ( Dump (check); )
#endif
                        check->left  = max (check->left,  left);       // use
smaller rect in H
                        check->right = min (check->right, right);
                        check->top   = min (check->top,   top);        // possibly
expand upwards
                        CDBG2 ( sdgOut << "After:\n"; Dump (check); )
                        return; // bottom stays the same
                    }
                }
                else if (top <= check->bottom)  // new FR is below & overlapping "check"
                {
                    check->left           = max (check->left,           left);
                    check->right    = min (check->right,    right);
                    check->bottom   = max (check->bottom,   bottom);       // possibly
expand downwards
                    CDBG2 ( sdgOut << "Merging two FR's #2" << endl; )
                    return;        // top stays the same
                }
            }

            if (abs (check->top    - top)    < kPercentage * height &&
                abs (check->bottom - bottom) < kPercentage * height)
            // Vertical dimension aligns; check for overlap in H
            {
```

```
            if (check->right >= right)
            {
                if (check->left <= right)// overlap in X? (new is to left of check,
but overlapping)
                {
                    check->top = max (check->top, top);     // use smaller rect
in V
                    check->bottom = min (check->bottom, bottom);
                    check->left = min (check->left, left);  // expand leftwards
                    CDBG2 ( sdgOut << "Merging two FR's #3" << endl; )
                    return; // right stays the same
                }
            }
            else if (left <= check->right)  // new FR is to right of "check" but
overlapping
            {
                check->top    = max (check->top,    top);
                check->bottom = min (check->bottom, bottom);
                check->right  = max (check->right, right);      // expand rigtwards
                CDBG2 ( sdgOut << "Merging two FR's #4" << endl; )
                return;         // left stays the same
            }
        }
    }

    if (m_skipSmall)
    {
        if ((right-left) < 20  ||  (bottom-top) < 20)
            return;
```

```
        if ((double)(right-left) * (double)(bottom-top) < 900)
            return; // an icon is 32x32
    }

    // Insert the new FR.
    DBG ( FrIter    pNew = )
    m_freeRects.insert (pInsertBefore, ccRect (left, top, right, bottom));

    CDBG0 ( sdgOut ("Created %X: (%ld..%ld, %ld..%ld)\n") << FrAddress (pNew) << left << right
<<top << bottom; )
} // AddFR()



/*
+=============================================================================================+
| DelFR
|
+=============================================================================================+
*/
void sdgFreeRect::DelFR (FrIter pFR)
{
    CDBG1 ( sdgOut ("DelFR: %X\n") << FrAddress (pFR); )
    m_freeRects.erase (pFR);
    CDBG2 ( sdgOut << "New list is:\n";  Dump(); )
}


/*
```

```
+=============================================================================================+
| RegisterOccupiedRectangle    "Apply" a screen object (represented by rectangle occupRect)   |
|                                  to the registered Free Rectangles.  Return
False iff infinite    |
|                                  loop detected (merely a precaution).
|
+=============================================================================================+
*/
bool sdgFreeRect::RegisterOccupiedRectangle (const ccRect &occupRect)
{
    ENTER1 ("RegisterOccupiedRectangle");
    CDBG1 ( sdgOut ("(%3ld..%3ld, %3ld..%3ld)\n") << occupRect.left << occupRect.right <<
occupRect.top << occupRect.bottom; )
    DBG ( m_numPasses++; )

    for (FrIter pCur = m_freeRects.begin();  pCur != m_freeRects.end();  )
    {
        FrIter  pNext = pCur;  pNext++; // squirrel value since pCur may get destroyed

        CDBG0 ( sdgOut ("Comparing occupied rect with FR %lX (%3ld..%3ld,%3ld..%3ld)\n") <<
                FrAddress (pCur) << pCur->left << pCur->right << pCur->top <<
pCur->bottom; )

        if (occupRect.right > pCur->left)
        {
            if (occupRect.left < pCur->right)       // overlap in X
            {
                if (occupRect.bottom > pCur->top)
                {
```

```
                        if (occupRect.top < pCur->bottom)       // overlap in Y:
bingo!
                        {
                            if (pCur->left < occupRect.left)
                                AddFR (pCur, pCur->left, pCur->top,
occupRect.left, pCur->bottom);
                            if (pCur->right > occupRect.right)
                                AddFR (pCur, occupRect.right, pCur->top,
pCur->right, pCur->bottom);
                            if (pCur->top < occupRect.top)
                                AddFR (pCur, pCur->left, pCur->top,
pCur->right, occupRect.top);
                            if (pCur->bottom > occupRect.bottom)
                                AddFR (pCur, pCur->left, occupRect.bottom,
pCur->right, pCur->bottom);
                            DelFR (pCur);
                        }
                    }
                }
            }
            pCur = pNext;
        } // pCur
        return true;
} // RegisterOccupiedRectangle()


//-------------------------------------------------------------------------------
//          Utility functions used by Reposition_Analytic()
```

```
//-----------------------------------------------------------------------------
inline long DistFromCenter_1_Dimension (long edge_1, long edge_2, long center = 0)
{
	ASSERT (edge_2 >= edge_1);
#if 0
	if (Within (center, edge_1, edge_2))
		return 0;
	return min (abs (edge_1 - center), abs (edge_2 - center));
#else
	if (edge_2 < center)
		return center - edge_2;
	if (edge_1 > center)
		return edge_1 - center;
	return 0;
#endif
}
//-----------------------------------------------------------------------------
inline long DistFromCenter (const ccRect &rect, long center_x = 0, long center_y = 0)
{
	long	dx = DistFromCenter_1_Dimension (rect.left, rect.right,  center_x),
			dy = DistFromCenter_1_Dimension (rect.top , rect.bottom, center_y);
	return dx * dx + dy * dy;
}
//-----------------------------------------------------------------------------
static ccRect	ScaleAndCenter (double width, double height, double scalingFactor)
{
	width  *= scalingFactor;
	height *= scalingFactor;
	ccRect  result;
```

```
    result.left   = -Round (width / 2);
    result.right  =  Round (width / 2);
    result.top    = -Round (height / 2);
    result.bottom =  Round (height / 2);
    return result;
}
//-----------------------------------------------------------------------------


/*
+=============================================================================================+
|
|
| Reposition    Place fragments on-screen and spaced apart after they are redrawn.
|
|
|
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|     Analytic repositioning is only applied if drawing de novo, since it destroys the
|
|     relative positions of molecules.  Dynamic repositioning is performed regardless.
|
+=============================================================================================+
*/
void SDG_Whole_PostProcessing::Reposition()
{
    if (PD.GetNFrags() <= 1)
        return;
```

```
    // Analytic Repositioning
    if (OpFlagged (kIgnoreCoordinates)  ||  OpFlagged (kReposition))
        Reposition_Analytic();

    // Dynamic Repositioning
    Reposition_Dynamic();

} // Reposition



/*
+=============================================================================================+
|
|
| Reposition_Analytic   The analytic repositioning procedure.
|
|
|
+---------------------------------------------+
|     ALGORITHM
|
|
|
|     1.     Rank fragments by decreasing size.
|
|     2.     In order of decreasing size:
|
```

```
|          a.    Find free rectangle that is closest to center (0,0) and large enough to
|
|                accommodate the fragment.
|
|          b.    Place the fragment there, as close as possible to the center.
|
|          c.    Recenter the fragments so they center on the origin (0,0).  Or
equivalently,  |
|                track the new central position, defined as the center of the smallest
bounding       |
|                rectangle of the placed fragments.
|
|
|
+=============================================================================================+
*/
void SDG_Whole_PostProcessing::Reposition_Analytic()
{
/*
+-------------------------------------------------+
|    Remap the molecular coordinate system to a nice, large integral coordinate space that  |
|    the Free Rectangle class can use.  The present molecules' scaling may be anything,
|
|    from very tiny exponentials to very large ones.  Moreover, different fragments might   |
|    in principle reside wildly far apart, or superimposed.  We do expect, however, that
|
|    they are all scaled similarly.  That is, if one molecule's bond lengths are about
|
|    7*10-3, then the other molecules' bonds fall in the same ballpark.
```

```
|
+-------------------------------------------------+
*/
    ENTER0 ("Reposition_Analytic");
    const double   avgBndLen = (NB == 0) ? (m_stdBondLen_W < 1.0E-12 ? 100. : m_stdBondLen_W)
: M.MedianBondLength (NULL, kIn2D);
    const long          kIntegralBondLength = 10;      // A nice, well-behaved integral
bond length
    const double   scalingFactor = kIntegralBondLength / avgBndLen;
    DBG ( if (0) )
    CDBG0 (  { sdgOut << "Before AnaRepo:\n"; DumpCoords (kAllFragments); sdgOut
("scalingFactor = %6.2lf\n") << scalingFactor; } )
    ccSet  sFragsToPlace (PD.GetNFrags());
    sFragsToPlace.Fill();
sFragsToPlace = m_FrgsToPlace;  // this line makes it compatible w/old behavior; remove this line
at some point -heh 7/27/99.

    // Rank by decreasing size
    multimap<long,int>     bySize;
    int                    frgNum;
    vector<ccRect>  frgDimensions (sFragsToPlace.Last() + 1);      // Integral fragment
position
    ccVec2D             maxTargetSpace (kIntegralBondLength, kIntegralBondLength);     //
Start off with some small number
    LOOP_SET (sFragsToPlace, frgNum)
    {
        double        dx, dy;
        M.GetSize (&dx, &dy, &PD.GetFragAtms (frgNum));
        ccRect        minmax = ScaleAndCenter (dx, dy, scalingFactor);
```

```
        minmax.InflateRect (kIntegralBondLength / 2, kIntegralBondLength / 2);  // add a
half-bond length margin all about each molecule
        const long      area = 4 * minmax.right * minmax.bottom;
        bySize.insert (pair<const long,int> (-area, frgNum));   // Use negative area to get
automatic sorting by decreasing size.
        frgDimensions [frgNum] = minmax;

        maxTargetSpace.x += dx * scalingFactor + kIntegralBondLength;
        maxTargetSpace.y += dy * scalingFactor + kIntegralBondLength;
    }

    const ccRect    targetSpace (-maxTargetSpace.x, -maxTargetSpace.y, maxTargetSpace.x,
maxTargetSpace.y);
    sdgFreeRect     FRs (targetSpace, false);
    ccRect  usedRect (0,0,0,0);     // Describes the limits of placed fragments.
    long    center_x = (targetSpace.left + targetSpace.right ) / 2, // Shorthand for the center
of usedRect.  Will shift as we feed fragments.
                center_y = (targetSpace.top  + targetSpace.bottom) / 2;

    for (multimap<long,int>::iterator  it = bySize.begin();  it != bySize.end();   )
    {
        const int       frgNum = it->second;
        ccRect          &curFragRect = frgDimensions [frgNum];
        const long      width  = curFragRect.Width(),
                        height = curFragRect.Height();

        CDBG1 ( sdgOut ("Placing fragment %ld (area %ld): width = %ld; height = %ld\n") <<
                                    frgNum << -it->first << width << height; )
```

```
        // Find free rectangle large enough and closest to center
        ccRect  best;
        bool    foundBest = false;
        for (sdgFreeRect::FrIter pFR = FRs.m_freeRects.begin();  pFR !=
FRs.m_freeRects.end();  pFR++)
        {
            if (pFR->Width() < width  ||  pFR->Height() < height)
                continue;
            if (!foundBest  ||  DistFromCenter (*pFR, center_x, center_y) <
DistFromCenter (best, center_x, center_y))
            {
                foundBest = true;
                best = *pFR;
            }
        }
        if (!foundBest)
        {
            ASSERT (false); // shouldn't happen
            best = *FRs.m_freeRects.begin();
        }
        CDBG1 ( sdgOut ("Free spot to place fragment is (%ld..%ld, %ld..%ld)\n") <<
best.left << best.right << best.top << best.bottom; )

        /*
        +----------------------------------------
+
        |   Translate the current fragment within the "best" rectangle so that it is as
|
        |   close to center as possible.  This involves locating the two sides closest
```

```
to    |
      |    the center.  For either dimension (vertical or horizontal), there are three
|
      |    cases:
|
      |         a.    "Best"'s Low value is closest to center.  Set fragment's
low value to   |
      |              this, and its hight value to Low + width/height.
|
      |         b.    "Best"'s High value is closest to center.  Set fragment's
high value to  |
      |              this, and its low value to High - width/height.
|
      |         c.    "Best"'s Low and High values flank the center.  Center
fragment's low |
      |              and high values on the center.
|
      +-----------------------------------------
+
      */
      if (best.left > center_x - width / 2)
      {
            curFragRect.left  = best.left;
            curFragRect.right = best.left + width;
      }
      else if (best.right < center_x + width / 2)
      {
            curFragRect.right = best.right;
            curFragRect.left  = best.right - width;
```

```
        }
        else
        {
            curFragRect.left  = center_x - width/2;
            curFragRect.right = center_x + width/2;
        }

        if (best.top > center_y - height / 2)
        {
            curFragRect.top    = best.top;
            curFragRect.bottom = best.top + height;
        }
        else if (best.bottom < center_y + height / 2)
        {
            curFragRect.bottom = best.bottom;
            curFragRect.top    = best.bottom - height;
        }
        else
        {
            curFragRect.top    = center_y - height/2;
            curFragRect.bottom = center_y + height/2;
        }
        CDBG1 ( sdgOut ("Fragment slid to position (%ld..%ld, %ld..%ld)\n") <<
curFragRect.left
                << curFragRect.right << curFragRect.top << curFragRect.bottom; )
        ASSERT (Within (curFragRect.left, best.left, best.right)  &&  Within
(curFragRect.right,  best.left, best.right));
        ASSERT (Within (curFragRect.top,  best.top,  best.bottom)  &&  Within
(curFragRect.bottom, best.top,  best.bottom));
```

```
        it++;

        // Insinuate the used rectangle on the free rectangle model.
        if (it != bySize.end()) // no point if this is the last fragment
            FRs.RegisterOccupiedRectangle (curFragRect);

        // Update the limits of the placement rectangle, and its center.
        usedRect |= curFragRect;
        center_x = (usedRect.left + usedRect.right)  / 2;
        center_y = (usedRect.top  + usedRect.bottom) / 2;
        CDBG1 ( sdgOut ("center is now (%d,%d)\n") << center_x << center_y; )
    }

#ifdef _DEBUG
    int    i,j;   // Ensure we succeeded in spacing the frags apart
    LOOP_SET (sFragsToPlace, i)
        LOOP_SET2 (sFragsToPlace, i, j)
            ASSERT (!frgDimensions [i].Intersects (frgDimensions [j]));
#endif

    // Translate the real fragments, preserving the current center.
    ccVec2D SER_min, SER_max;
    ccSet   asAllAtoms (NA); asAllAtoms.Fill();
    SmallestBoundingRect (SER_min, SER_max, asAllAtoms);
    ccPoint2D      cent ((SER_min.x + SER_max.x) / 2,
                         (SER_min.y + SER_max.y) / 2);
    ccVec2D center_offset (cent.x - center_x / scalingFactor,
                           cent.y - center_y / scalingFactor);
    CDBG1 ( sdgOut ("\nBeginning translation.  center_offset_x/_y = (%lf,%lf)\n") <<
```

```
center_offset.x << center_offset.y; )
      LOOP_SET (sFragsToPlace, frgNum)
      {
            ccVec2D              oldFrag_min, oldFrag_max;
            SmallestBoundingRect (oldFrag_min, oldFrag_max, PD.GetFragAtms (frgNum));
            const ccRect    &curFragRect = frgDimensions [frgNum];
            const ccPoint2D topLeftPt ((curFragRect.left + kIntegralBondLength/2) /
scalingFactor,
                                                (curFragRect.top  +
kIntegralBondLength/2) / scalingFactor);
            const ccVec2D   dxy = topLeftPt - oldFrag_min;
            // topLeftPt is the point to which we wish to translate the top left corner.
            ccTranslate (M, dxy.x + center_offset.x, dxy.y + center_offset.y, 0.,
&PD.GetFragAtms (frgNum));
            CDBG2 ( sdgOut ("After translating fragment %d:\n") << frgNum;  DumpCoords
(kAllFragments); )
      }
      if (s_dbgFlags.GR_tracing)      { sdgOut << "Reposition_Analytic: Ending molecule is:\n";
DumpCoords (kAllFragments); }

}
```